Scott Cain

2/23/2025

CS-320 Project Two

Professor Lopez

**Summary and Reflections Report**

For the Contact Service, Task Service, and Appointment Service, my testing approach was using JUnit and coverage testing. Using these techniques was a great way to check the software met the necessary standards while highlighting areas for improvement. Unit testing with JUnit verified that individual units or components of the software performed as I intended for them. For example, the contact service milestone, I developed unit tests for adding, updating, and deleting contacts. These tests aligned well with the software requirements.

I defended the overall quality of my JUnit tests by reflecting on the coverage percentage. Using the EclEmma plugin, I have a 84.6% coverage for the Appointment Service, this means that I have covered the code needed. The contact service had a lower coverage of 37.9% first time I made it but it did highlight areas for improvement and later on I was able to improve it over 90%.

My experience writing JUnit tests was a great way to improve my skills. It let me know my code was technically sound by developing tests for each function and using assertions to validate expected outcomes. For example, the contact service tests, I checked for null values and handled exceptions correctly. Efficient coding was developed by avoiding any redundancy and optimizing test cases for the best performance possible.

In this project, I used several software testing techniques, that include unit testing, integration testing, and coverage testing. Unit testing helped catch bugs and code reliability early. Integration testing validated the data flow and interactions between modules. This leads to better coverall functionality and robustness of the task and appointment services. Coverage testing provided a measure of how much of the source code was tested, with higher percentages indicating more thorough testing. This was something I missed to begin with but was able to improve myself with later.

There were other testing techniques that I did not use in this project, such as system testing and acceptance testing. System testing validates the complete and integrated software product against the specified requirements. Although I didn't use this technique, it checks that entire system functions as intended. Acceptance testing determines whether the system meets business requirements and is typically used by stakeholders or end-users. This technique ensures the software meets user needs and expectations before going live.

Each testing technique has practical uses and implications for different software development projects and situations. Unit testing reduces debugging time and improves code quality. Integration testing catches interface issues early, ensuring smooth project execution. System testing validates the entire system's behavior before release. With acceptance testing it helps increases user satisfaction and reduces post-release issues.

While working on this project, I adopted an adaptive mindset. Having the mindset to adapt let me review and receive feedback to change my coding. I also feel that I have a mindset for finding errors. For example, I tested edge cases and potential failure points to know my code has a robustness to not slow down the program.

Being disciplined in my commitment to quality as a software engineering professional is always key. Cutting corners can lead to technical debt, which I will avoid by adhering to best practices and thorough testing. By maintaining high standards and continually learning, I plan to ensure the long-term success and maintainability of my code.